EX : MLFLOW

Do it as a team (2000 points)

Part 1: MLflow Tracking

Objective: Understand and implement MLflow tracking in a collaborative project.

1. **Task A:** Set up an MLflow server locally or on a cloud platform.
    a. Assign two team members to configure and run an MLflow tracking server.
    b. Ensure the server can store experiment data and artifacts.
2. **Task B:** Integrate MLflow tracking into an existing machine learning training script.
    a. Assign two team members to work on modifying the training script to log:
        i. Parameters (e.g., learning rate, batch size).
        ii. Metrics (e.g., accuracy, loss).
        iii. Model artifacts (e.g., trained models).
        iv. System information (e.g., runtime, hardware used).
3. **Task C:** Visualize experiment results.
    a. Assign one team member to analyze the logged metrics and artifacts using MLflow UI.
    b. Create a comparison report of different runs.

Part 2: Model Registry

Objective: Manage models through the MLflow model registry.

1. **Task A:** Register a model in MLflow.
    a. Assign one team member to select the best model from Part 1 experiments and register it in the model registry.
2. **Task B:** Transition model stages.
    a. Assign one team member to change the model stage from "Staging" to "Production" based on predefined criteria (e.g., accuracy above a threshold).

3. **Task C:** Manage multiple models.
    a. Assign two team members to work on versioning. They will compare two versions of the model in terms of performance and maintain a log of their observations.
4. **Task D:** Document the model lifecycle.
    a. Assign one team member to write detailed documentation on the lifecycle of the registered models, including their criteria for promoting models to production.

Part 3: Make Predictions

Objective: Deploy and test the registered model for inference.

1. **Task A:** Set up an inference environment.
    a. Assign two team members to deploy the "Production" model as a REST API using Flask or FastAPI.
2. **Task B:** Write a client application.
    a. Assign two team members to create a Python script that sends test data to the deployed model's endpoint and displays predictions.
3. **Task C:** Test and evaluate the deployment.
    a. Assign one team member to:
        i. Create a test dataset.
        ii. Test the model's predictions for accuracy and latency.
        iii. Report issues and suggest improvements.

EX : CI/CD

Building a Machine Learning Flask Application with CI/CD and Cloud Deployment

(3000 points)

Objective:

By the end of this exercise, students will:

1. Develop a Flask application that integrates a machine learning model for prediction.
2. Set up a CI/CD pipeline using GitHub Actions.
3. Deploy the application on a cloud platform (e.g., AWS, Azure, or GCP).

Team Roles and Responsibilities:

1. **Backend Developer (Member 1)**
    a. Develop the Flask application structure.
    b. Create endpoints for input, prediction, and result display.
2. **Model Developer (Member 2)**
    a. Train or use a pre-trained machine learning model.
    b. Integrate the model into the Flask application.
3. **CI/CD Engineer (Member 3)**
    a. Set up a GitHub repository and create a GitHub Actions workflow for CI/CD.
    b. Automate testing, building, and deployment steps.
4. **Cloud Deployment Specialist (Member 4)**
    a. Choose a cloud provider.
    b. Set up the infrastructure (e.g., virtual machine, container service) and deploy the application.
5. **Project Manager & QA Engineer (Member 5)**
    a. Coordinate team efforts, track progress, and ensure deadlines are met.
    b. Test the deployed application and provide feedback for improvement.

Tasks

Phase 1: Develop the Flask Application

1. **Backend Developer:**
    a. Set up a Flask project folder structure (app.py, templates/, static/, etc.).
    b. Develop a home page and prediction endpoint (/predict).
2. **Model Developer:**
    a. Choose a dataset (e.g., Iris dataset or Boston Housing Prices).
    b. Train a model using scikit-learn or TensorFlow.
    c. Save the model in a deployable format (.pkl or .h5).
    d. Write a script to load the model and use it for predictions.

Phase 2: Implement CI/CD Using GitHub Actions

1. **CI/CD Engineer:**
    a. Set up a GitHub repository and add the Flask application.
    b. Write a GitHub Actions workflow file (.github/workflows/main.yml) that includes:
        i. Installing dependencies (Python, Flask, etc.).
        ii. Running tests for the Flask app.
        iii. Building and packaging the application (e.g., Docker container).
        iv. Deploying to a cloud service.
2. **Project Manager & QA Engineer:**
    a. Test the CI/CD pipeline for correctness.
    b. Ensure the pipeline automatically triggers on push or pull request to the main branch.

Phase 3: Deploy on the Cloud

1. **Cloud Deployment Specialist:**
    a. Select a cloud provider (AWS, Azure, or GCP), , digital ocean, heroku , ...
    b. Set up a virtual machine, container service (like AWS ECS), or serverless platform
    c. Configure the environment to host the Flask application.
    d. Use GitHub Actions to deploy the application to the cloud.
2. **Backend Developer & Model Developer:**
    a. Assist with resolving any cloud-specific deployment issues.

Phase 4: Testing and Feedback

1. **Project Manager & QA Engineer:**
    a. Test the deployed application using sample inputs.
    b. Ensure endpoints work as expected and monitor response times.
    c. Report any bugs or performance issues to the team.